Aufgabe V6: Semaphore im Einsatz

Eine Pufferverwaltung auf Anwendungsebene erlaubt es, Puffer fester Größe anzufordern und wieder freizugeben. Die Anzahl der Puffer wird einmalig bei der Initialisierung festgelegt. Die Puffer sind ab 0 durchnummeriert. Die Verwaltung verwendet einen Bitvektor entsprechender Länge, in dem 0 einen freien und 1 einen belegten Puffer anzeigt. Die Operationen laufen wie folgt ab:

Anfordern: *kein Argument, Rückgabewert ist eine Puffernummer oder ein Fehler*

1. im Bitvektor nach einer 0 suchen
2. nichts gefunden ⇒ Fehler zurückgeben
3. gefundenes Bit auf 1 setzen
4. entsprechende Puffernummer zurückgeben

Freigeben: *Argument ist eine Puffernummer, kein Rückgabewert*

8. im Bitvektor das entsprechende Bit auf 0 setzen

Die Pufferverwaltung ist noch nicht für parallele Aufrufe durch mehrere Prozesse (engl.: *threads*) gleichzeitig ausgelegt. Gehen Sie davon aus, dass die Anwendung nur gültige Aufrufe durchführt, also nur zuvor angeforderte Puffer freigibt.

a) Bestimmen Sie die kritischen Abschnitte beider Operationen.

Die kritischen Abschnitte sind die Zugriffe auf den gemeinsamen Bitvektor. Beim **Anfordern** also Schritt 1 und gegebenfalls Schritt 3. Die Abfrage in Schritt 2 muss ebenfalls innerhalb des kritischen Abschnitts stattfinden, damit die Schritte 1 und 3 ungestört hintereinander ablaufen. Das Zurückgeben des Fehlers in Schritt 2 sowie der Puffernummer in Schritt 4 gehören nicht zum kritischen Abschnitt.

Beim **Freigeben** ist der gesamte Schritt 8 ein kritischer Abschnitt.

b) Sichern Sie die Pufferverwaltung mit einem Semaphor `M` so ab, dass höchstens ein kritischer Abschnitt gleichzeitig ablaufen kann. Mit welchem Wert wird `M` initialisiert? Wo müssen Aufrufe von `M.P()` bzw. `M.V()` stattfinden?

`M` wird mit 1 initialisiert, denn höchstens *ein* kritischer Abschnitt darf ablaufen. Ein Aufruf von `M.P()` sichert exklusiven Zugriff auf den Bitvektor, `M.V()` gibt den Zugriff wieder frei.

Anfordern: `M.P()` vor Schritt 1, `M.V()` nach Schritt 3 und nach ⇒ in Schritt 2

Freigeben: `M.P()` vor Schritt 1, `M.V()` nach Schritt 1.

c) Ändern Sie die Pufferverwaltung mit einem zweiten Semaphor `Z` so, dass das Anfordern auf einen freien Puffer wartet, statt einen Fehler zurückzugeben. Mit welchem Wert wird `Z` initialisiert? Wo müssen Aufrufe von `Z.P()` bzw. `Z.V()` stattfinden?

`Z` wird mit der Anzahl der (freien) Puffer initialisiert. Anfordern eines Puffers zählt das Semaphor runter, Freigeben wieder rauf.

Beim Anfordern wird `Z.P()` vor Schritt 1 und vor `M.P()` aufgerufen. Ein Zugriff auf den Bitvektor erfolgt also nur, wenn ein Puffer frei ist. Schritt 2 entfällt damit.

Beim Freigeben wird `Z.V()` irgendwann nach `M.P()` aufgerufen. Wann genau ist für die Logik nicht wichtig, da der Bitvektor bis `M.V()` gesperrt bleibt. Es ist aber guter Stil, zuerst mit `M.V()` die Exklusivsperre freizugeben, dann erst mit `Z.V()` den Zähler zu aktualisieren.

d) Was kann schiefgehen, wenn zwei Prozesse gleichzeitig die ungesicherte Pufferverwaltung nutzen, also ohne das Semaphor `M`?

- beide rufen **Anfordern**
- beide rufen **Freigeben**
- einer ruft **Anfordern**, der andere **Freigeben**

In beiden kritischen Abschnitten liest ein Prozess den Bitvektor, ändert ein Bit und schreibt den geänderten Bitvektor zurück in den Speicher. Fehler entstehen, wenn zwischen Lesen und Schreiben ein anderer Prozess den Bitvektor liest und mit dem gleichen, alten Wert arbeitet.

Anfordern x2: Beide Prozesse finden und setzen die gleiche 0 im Bitvektor. Anschließend verwenden sie den gleichen Puffer und verfälschen gegenseitig ihre Daten darin. Falls die Prozesse dadurch nicht abstürzen wird der Puffer später zweimal freigegeben, was zu weiteren Problemen führen könnte.

Freigeben x2: Beide setzen eine andere 1 im Bitvektor auf 0, aber eine dieser Änderungen wird von der anderen überschrieben. Dadurch bleibt einer der freigegebenen Puffer als belegt markiert und ist nicht mehr nutzbar.

Anfordern und Freigeben: Entweder des Setzen oder das Löschen eines Bits wird überschrieben. Im ersten Fall bleibt der gerade belegte und nun verwendete Puffer als frei markiert und kann von einem weiteren Prozess belegt werden. Im zweiten Fall bleibt ein freigegebener Puffer als belegt markiert und ist nicht mehr nutzbar.